AMENDMENTS TO THE CLAIMS:

This listing of claims will replace all prior versions, and listings, of claims in the application:

1. (Original) A method of enabling multiple different operating systems to run concurrently on the same computer, comprising:

selecting a first operating system to have a relatively high priority;

selecting at least one second operating system to have a relatively lower priority;

providing a common program arranged to switch between said operating systems under predetermined conditions; and

providing modifications to said first and second operating systems to allow them to be controlled by said common program.

2. (Original) The method of claim 1, in which the first operating system is a real time operating system.

3. (Original) The method of claim 1, in which the second operating system is a non-real time, general-purpose operating system.

4. (Original) The method of claim 1, in which the second operating system is Linux, or a version or variant thereof.

5. (Original) The method of claim 1, in which the common program is arranged to save, and to restore from a saved version, the processor state required to switch

between the operating systems.

6. (Original) The method of claim 1, in which processor exceptions for the second operating system are handled in virtual fashion by the common program.

7. (Original) The method of claim 1, in which the common program is arranged to intercept some processor exceptions, and to call exception handling routines of the first operating system to service them.

8. (Original) The method of claim 7, in which the processor exceptions for the second operating system are notified as virtual exceptions.

9. (Original) The method of claim 8, in which the common program is arranged to call an exception handling routine of the second operating system corresponding to a said virtual exception which is pending.

10. (Original) The method of claim 1, further comprising providing each of said operating systems with separate memory spaces in which each can exclusively operate.

11. (Original) The method of claim 1, further comprising providing each of said operating systems with first input and/or output devices of said computer to which each has exclusive access.

12. (Original) The method of claim 11, in which each operating system accesses said first input and/or output devices using substantially unmodified native routines.

13. (Original) The method of claim 1, further comprising providing each of said operating systems with access to second input and/or output devices of said computer to which each has shared access.

14. (Original) The method of claim 13, in which all operating systems access said second input and/or output devices using the routines of the first operating system.

15. (Original) The method of claim 1, further comprising providing a restart routine for restarting a said second operating systems without interrupting operation of said first, or said common program.

16. (Original) The method of claim 15, in which the common program provides trap call mechanisms, to control the operation of the second operating system, and/or event mechanisms to notify the first operating system of status changes in the second operating system.

17. (Original) The method of claim 15, in which the common program stores a copy of the system image of the kernel of the second operating system, and is arranged

to restore the kernel of the second operating system from such a saved copy.

18. (Original) The method of claim 15, in which the first and second operating systems have cooperating routines to enable the first operating system to monitor the continued operation of the second operating system, to allow the detection of a crash of the second operating system.

19. (Original) The method of claim 1, further comprising providing a debug routine, in which the common program is arranged to output the states of machine state variables on occurrence of predefined conditions in the operation of said operating systems.

20. (Original) The method of claim 1, further comprising combining said operating systems and common program into a single code product.

21. (Original) The method of claim 1, further comprising embedding said operating systems and common program onto persistent memory on a computer product.

22. (Original) The method of claim 1, in which the common program is arranged to provide an inter-operating system communications mechanism allowing communications between said first and second operating systems, and/or applications running on them.

23. (Original) The method of claim 22, in which the common program defines virtual input and/or output devices corresponding to communications bus bridges, so that said operating systems can communicate as if by a communications bus.

24. (Original) The method of claim 23, in which the step of modifying said operating systems comprises adding driver routines managing said virtual bus bridge devices.

25. (Original) A development kit computer program product comprising code for performing the steps of claim 1.

26. (Original) A computer program product comprising code combined according to claim 20.

27. (Original) An embedded computer system comprising a CPU, memory devices and input/output devices, having stored on persistent memory therein programs embedded according to claim 24.

28. (Original) A computer system comprising a CPU, memory devices and input/output devices, having executing thereon computer code comprising;

a first operating system having a relatively high priority;

a second operating system having a relatively lower priority; and

a common program arranged to run said operating systems concurrently by switching between said operating systems under predetermined conditions.

29. (Currently Amended) A computer system according to claim 28, arranged to run said first and second operating systems concurrently ~~using the method of any of claims 1 to 24~~.

30. (Original) The method of claim 1, in which each said operating system is provided with an idle routine, in which it passes control to the common program.

31. (Original) The method of claim 30, in which said idle routine substitutes for a processor halt instruction.

32. (Original) The method of claim 1, in which, on occurrence of processor exception during execution of an executing operating system,

(a) the common program is arranged to call exception handling routines of the first operating system to service them;

(b) if the exception was intended for a predetermined second operating system, a virtual exception is created;

(c) after the processor exception has been serviced by the first operating system, the common program is arranged to return to execution of the executing operating system;

(d) when the common program next switches to the predetermined second operating system, the virtual exception which is pending is notified to the predetermined second operating system; and

an exception handling routine of the predetermined second operating system corresponding to the said virtual exception is called to service it.

33. (Original) The method of claim 1, in which the second operating system is modified to prevent it masking interrupts.

34. (Original) The method of claim 1, in which all hardware interrupts are initially handled by the first operating system, and those intended for a second operating system are virtualised and deferred until that second operating system is next scheduled by the common program, and are serviced by that second operating system at that time.

35. (Original) The method of claim 8, in which the common program is arranged to provide a means for the or each secondary operating system to mask virtual exceptions to replace the hardware interrupt masking code in the secondary operating system to make the secondary system fully preemptable by the primary system.

36. (Original) The method of claim 9, in which said second virtual exception is not masked.